Air Gap = Ambiente Frío (Fuera de línea)

Hot Node = Nodo Sincronizado (En línea)

Cardano-cli:~$ Study sheets (Español)

Parte 1: llaves, direcciones y delegación

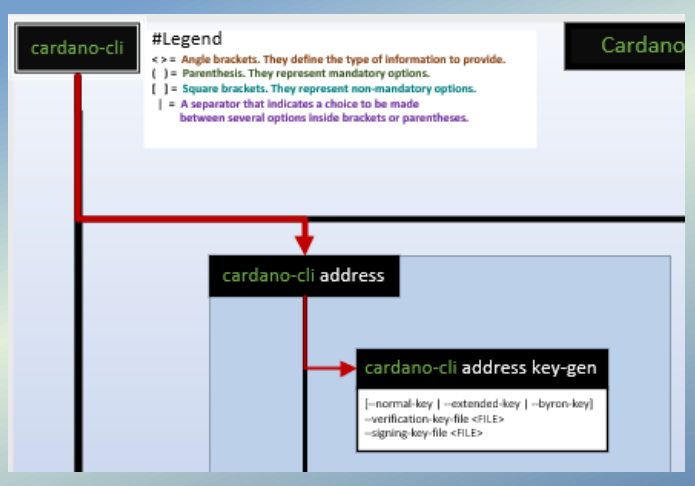
Traducido por Token Stake Pool

Este tutorial está diseñado para su uso en versión Imprimible de cardano-cli cheat sheet V8.0.0

Este documento tiene como meta explicar a detalle cómo interpretar los comandos en cardano-cli y sus opciones para poder ensamblarlos uno mismo si es necesario. Para esto, se requiere una computadora e instalar el Nodo de la cadena de bloques de Cardano y la interfaz por línea de comandos (cardano-cli). Se comienza con comandos simples y se incrementa la complejidad conforme progresa el tutorial.

Primer ejercicio: Creación de llaves de pago y de participación (stake) — Air Gap

1 **Primero, localiza la ramificación que vas a usar para tus llaves de pago.**



2 **Hay 5 opciones en total.**

Las primeras 3 opciones tienen corchetes con 2 separadores que indican que es opcional escoger entre estas 3 opciones, ya que la llave normal se usará por defecto si no se especifica una. Para este ejemplo, no se utilizarán.

```
cardano-cli address key-gen
[--normal-key | --extended-key | --byron-key]
--verification-key-file <FILE>
--signing-key-file <FILE>
```

3 **Se deben usar las siguientes 2 opciones.**

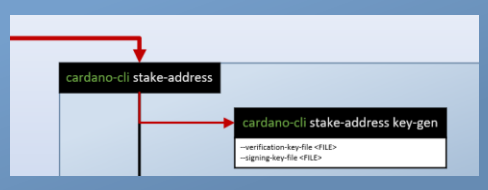
Los paréntesis angulares indican el tipo de información <File>. Se debe proporcionar el nombre asignado a los dos archivos con las respectivas llaves.

```
cardano-cli address key-gen
[--normal-key | --extended-key | --byron-key]
--verification-key-file payment.vkey
--signing-key-file payment.skey
```

4 **Este es el resultado final de este simple comando en tu terminal.**

```
user@computer:~$ cardano-cli address key-gen \
> --verification-key-file payment.vkey \
> --signing-key-file payment.skey
```

5 **Ahora que las llaves de pago están listas, debes crear las llaves de participación (stake) . Esta es incluso más fácil porque sólo existe un tipo de esta llave de delegación.**




```
cardano-cli stake-address key-gen
--verification-key-file <FILE>
--signing-key-file <FILE>
```

6 **Igual que 3. Debes indicar el tipo de información. <File>**

```
cardano-cli stake-address key-gen
--verification-key-file stake.vkey
--signing-key-file stake.skey
```

7 **Y para el resultado final en la terminal …**

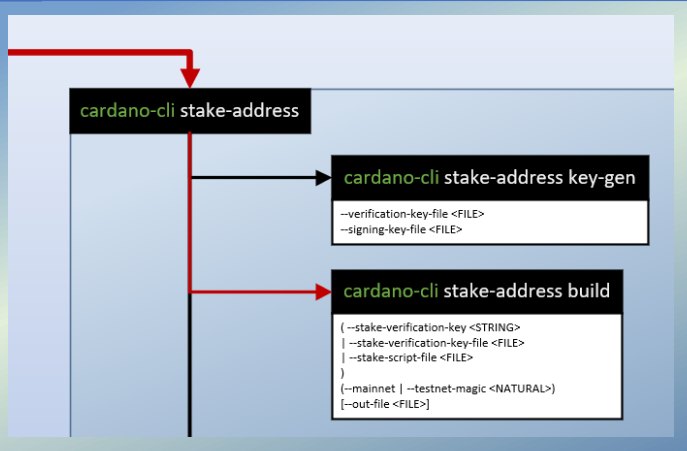
```
user@computer:~$ cardano-cli stake-address key-gen \
> --verification-key-file stake.vkey \
> --signing-key-file stake.skey
```

Advertencia. Se recomienda generar y usar sus llaves de pago (payment) y de delegación (stake) para firmar transacciones en un ambiente frío (Air Gap) por motivos de seguridad. https://developers.cardano.org/docs/get-started/air-gap

Ahora que los 2 pares de llaves fueron creados, puedes crear una dirección de delegación (stake) que te permitirá consultar la cantidad de recompensas y retirarlas al hacer una transacción con tu stake.skey

Segundo ejercicio: Creación de una dirección de stake — Air Gap

1 **Localizar la ramificación que usarás para crear tu dirección de stake.**



2 **Hay un total de 6 opciones.**

Las primeras 3 opciones están unidas por paréntesis y tienen 2 separadores que indican una elección obligatoria entre alguna de estas tres opciones.

```
cardano-cli stake-address build
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
)
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 **Usarás la opción stake-verification-key-file.**

Los paréntesis angulares indican el tipo de información <File>. Esta vez, debes proporcionar la ruta tu archivo stake.vkey

```
cardano-cli stake-address build
( --stake-verification-key <STRING>
| --stake-verification-key-file stake.vkey
| --stake-script-file <FILE>
)
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

4 **Ahora tienes 2 opciones en paréntesis**

Debes especificar la red utilizada y si es testnet, mencionar el número de magia para la red. En lugar, usemos mainnet.

```
cardano-cli stake-address build
--stake-verification-key-file stake.vkey
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

5 **Y para la última opción –out-file <FILE>**

Esta no es obligatoria porque está dentro de corchetes. Pero si no usas esta opción, la salida (dirección de stake) del comando será desplegada en tu terminal, en vez de guardarse en un archivo, y ya que se va a necesitar después, hay que asignarle un nombre.

```
cardano-cli stake-address build
--stake-verification-key-file stake.vkey
--mainnet
--out-file stake.addr
```

6 **Así es como este comando buscará en tu terminal.**

```
user@computer:~$ cardano-cli stake-address build \
> --stake-verification-key-file stake.vkey \
> --mainnet \
> --out-file stake.addr
```

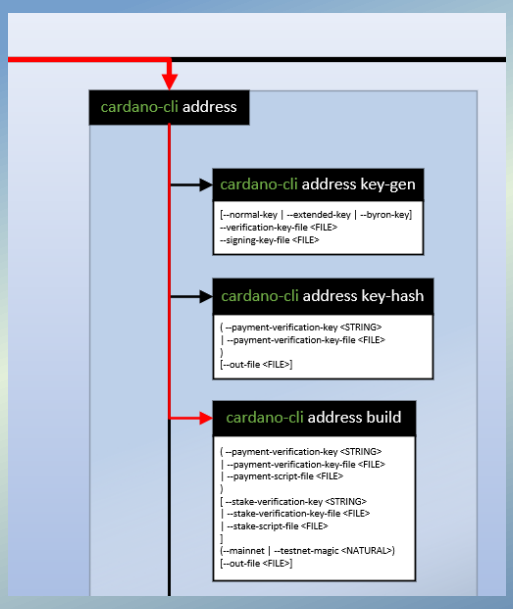
7 **Esto es lo que debes tener hasta ahora.**

```
user@computer:~$ ls
payment.vkey    payment.skey    stake.vkey
stake.skey      stake.addr
```

Ahora que los 2 pares de llaves y la dirección de stake fueron creados, podrás generar una dirección combinando tus llaves de pago y stake de forma que el dinero en la dirección generada se incluya en el protocolo de stake con tus recompensas.

Tercer ejercicio: Creación de archivo de pago con dirección de stake — Air Gap

1 **Localiza la ramificación que usarás para tu archivo de pago con dirección de stake.**



2 **Hay 9 opciones en total.**

Es posible crear una dirección de pago usando solamente tu llave de pago sin ligar a tu llave de stake, lo que explica por qué el 1er grupo de opciones es obligatorio y el 2ndo no.

cardano-cli address build

```
( --payment-verification-key <STRING>
| --payment-verification-key-file <FILE>
| --payment-script-file <FILE>
)
[ --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 **Hay que usar ambas llaves de verificación, de pago y de stake.**

Nuevamente, de acuerdo a los paréntesis angulares <FILE>, debes definir la ruta a ambos payment.vkey y stake.vkey.

cardano-cli address build

```
( --payment-verification-key <STRING>
| --payment-verification-key-file payment.vkey
| --payment-script-file <FILE>
)
[ --stake-verification-key <STRING>
| --stake-verification-key-file stake.vkey
| --stake-script-file <FILE>
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

4 **Ahora vas a mencionar la red a usar y el nombre del archivo de tu dirección.**

cardano-cli address build

```
--payment-verification-key-file payment.vkey
--stake-verification-key-file stake.vkey
(--mainnet | --testnet-magic <NATURAL>)
--out-file paymentwithstake.addr
```

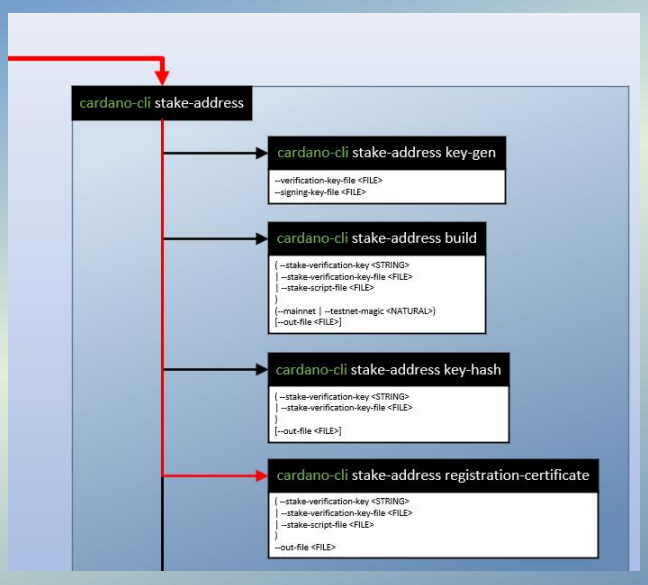
5 **Este es el resultado final.**

```
user@computer:~$ cardano-cli address build \
> --payment-verification-key-file payment.vkey \
> --stake-verification-key-file stake.vkey \
> --mainnet \
> --out-file paymentwithstake.addr
```

Puedes copiar el contenido de paymentwithstake.addr en un editor de texto y pegarla en una transacción dentro de la cartera de Cardano que vas a usar generalmente y enviar ADA a ella. (10 ADA deben ser suficientes para empezar)

Cuarto ejercicio: Creación del certificado de stake — Air Gap

1 **Localiza la ramificación que debes usar para tu certificado de delegación (stake)**



2 **Hay 4 opciones**

Para participar en el protocolo y delegar tu ADA, debes ligar tu llave de verificación de stake a un certificado que vas a someter a la cadena de bloques en los siguientes ejercicios. El comando para crear tu certificado es bastante simple. Sólo debes proporcionar uno de estos 3 ajustes obligatorios y especificar el nombre del archivo que servirá de certificado.

cardano-cli stake-address registration-certificate

```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
)
--out-file <FILE>
```

cardano-cli stake-address registration-certificate

```
( --stake-verification-key <STRING>
| --stake-verification-key-file stake.vrf
| --stake-script-file <FILE>
)
--out-file stake.cert
```

3 **Este es el resultado final de cómo se debe ver el comando en tu terminal.**

```
user@computer:~$ cardano-cli stake-address registration-certificate \
> --stake-verification-key-file stake.vkey \
> --out-file stake.cert
```

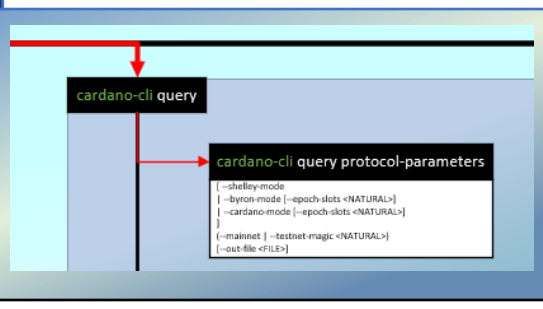
4 **Esto es lo que debes tener hasta ahora.**

```
user@computer:~$ ls
payment.vkey        payment.skey        stake.vkey        stake.skey
stake.addr          paymentwithstake.addr   stake.cert
```

Ahora vas a obtener los parámetros del protocolo y la punta de la cadena de bloques para empezar a construir tu primera transacción.

Quinto ejercicio: Obteniendo los parámetros del protocolo — Hot Node

1 **Primero, para tu transacción, necesitas los parámetros del protocolo para calcular la comisión.**



2 **Tienes 6 opciones y 2 subopciones en total.**

cardano-cli query protocol-parameters

```
[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 **Salta los modos de opciones. Menciona la red deseada y el nombre del archivo a crear.**

cardano-cli query protocol-parameters

```
[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file protocol.json
```

4 Este es el resultado final de cómo este comando debe verse en la terminal.

```
user@computer:~$ cardano-cli query protocol-parameters \
> --mainnet \
> --out-file protocol.json
```

5 Ahora avanza y visualiza el contenido de ese archivo:

```
user@computer:~$ cat protocol.json
```

En el archivo protocol.json busca el depósito a realizar en la cadena de bloques para registrar tu dirección de stake y participar en el protocolo de stake.

6 Anota el monto del depósito, ya que es necesario más adelante. La cantidad es en Lovelace. (1 ADA = 1,000,000 Lovelace)

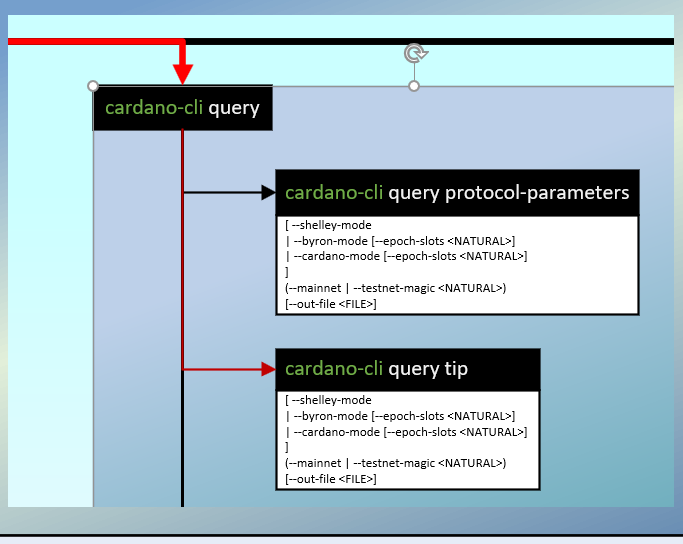
```
"poolRetireMaxEpoch": 18,
"protocolVersion": {
    "major": 8,
    "minor": 0
},
"stakeAddressDeposit": 2000000,
"stakePoolDeposit": 500000000,
"stakePoolTargetNum": 500,
"treasuryCut": 0.2,
"txFeeFixed": 155381,
"txFeePerByte": 44,
"utxoCostPerByte": 4310,
"utxoCostPerWord": null
```

7 Ahora debes tomar el archivo protocol.json y transferir a tu entorno frío "Air Gap" para poder calcular las comisiones al construir tus transacciones.

⚠ **Con CIP-1694 y la era Voltaire que está a la vuelta de la esquina, será posible para los propietarios de ADA en la comunidad, con la ayuda del comité constitucional y los DReps, modificar los parámetros del protocolo usando un sistema de votación bien desarrollado. Es por esto que es importante que tengas las modificaciones más recientes de estos protocolos en tu ambiente frío "Air Gap" ya que esto puede tener un impacto directo en varios parámetros relacionados con tus transacciones.**

Sexto ejercicio: Obtener la punta actual del nodo — Hot Node

1 En el siguiente ejercicio necesitas conocer la punta actual del nodo para calcular el TTL (descripción en siguiente ejercicio)



2 Así como en el ejercicio previo, 6 opciones y 2 subopciones.

Ahora que comienzas a entender completamente el principio puedes saltar varios pasos. No hay necesidad de crear un archivo, sólo se requiere el número de slot.

```
cardano-cli query tip
[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 Este es el resultado final en tu terminal.

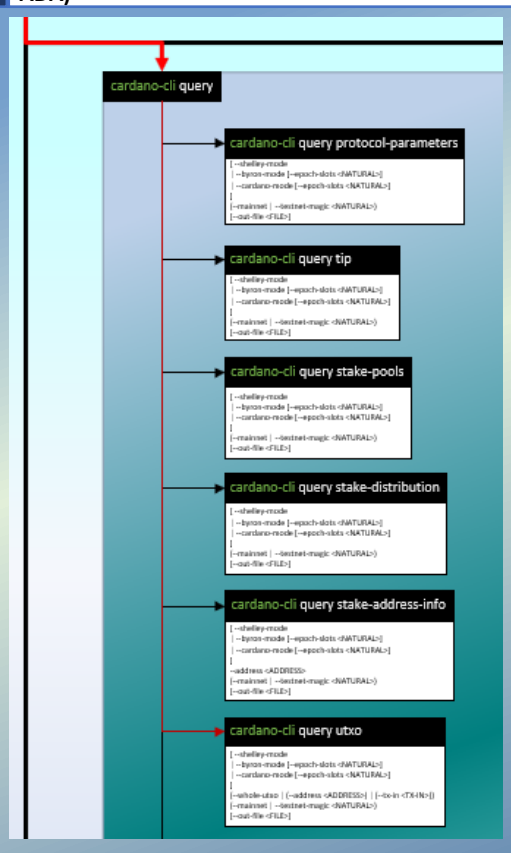
```
user@computer:~$ cardano-cli query tip \
> --mainnet
```

4 Anota el número de slot

```
{
    "block": 8749125,
    "epoch": 410,
    "era": "Babbage",
    "hash": "503e4af96abc18e1d4b5de08e0d35cb508e364...",
    "slot": 92027764,
    "syncProgress": "100.00"
}
```

Séptimo ejercicio: Consulta el UTXO — Hot Node

1 Ahora vas a consultar los UTXOs de tu paymentwithstake.addr (Si ya le habías mandado ADA)



2 Este comando tiene 9 opciones y 2 subopciones.

Debes consumir al menos una UTXO como entrada de la transacción. Una transacción contiene varias entradas y salidas pero en este caso debes tener sólo una UTXO asociada a tu paymentwithstake.addr porque solamente hiciste un depósito de 10 ADA. Por lo tanto, hay que usar sólo lo obligatorio. En resumen, tu paymentwithstake.addr, la red a usar y crear un archivo para llevar una lista de UTXOs al ambiente frío.

```
cardano-cli query utxo
[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--whole-utxo | (--address <ADDRESS>) | (--tx-in <TX-IN>))
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 Así se debe ver en tu terminal

```
cardano-cli query utxo
[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--whole-utxo | (--address paymentwithstake.addr) | (--tx-in <TX-IN>))
(--mainnet | --testnet-magic <NATURAL>)
(--out-file UTXO.addrs)
```

```
user@computer:~$ cardano-cli query utxo \
> --address paymentwithstake.addr
> --mainnet
> --out-file utxo.addrs
```

4 El contenido del archivo utxo.addr debe verse así. El UTXO de tu depósito de 10 ADA = 10,000,000 Lovelace. Ahora puedes tomar este archivo y copiarlo a tu ambiente frío "Air Gap". Lo vas a necesitar pronto.

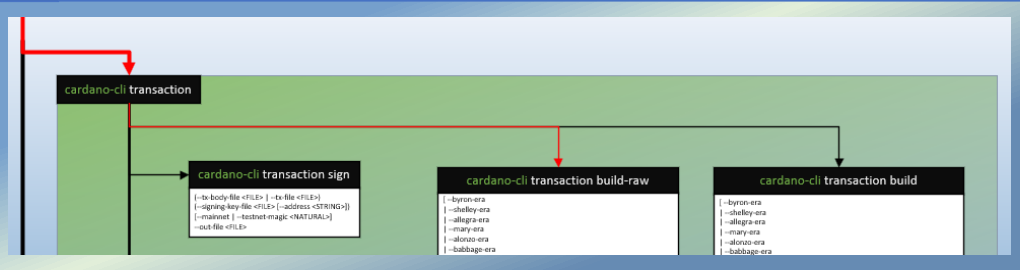
```
                           TxHash                                 TxIx        Amount
--------------------------------------------------------------------------------------
1234a4d18e9dkhb34234kjbvdec3ad81e299c1a523443453561e61ce9bf8608e8c802df3b7f8c     0        10000000 lovelace + TxOutDatumNone
```

Es tiempo de construir tu primera transacción, que se usará para someter un certificado de stake. Antes de comenzar, lo siguiente podrá parecer desalentador, pero al ir paso por paso se debe entender el porqué y cómo vas a reducir las siguientes opciones a 6 en total para el proceso de tu transacción. Por motivos de seguridad, en este tutorial usamos métodos que involucran el comando "cardano-cli transaction build-raw" en lugar de "cardano-cli transaction build" porque se puede construir en un entorno fuera de línea.

Octavo ejercicio: Creación del borrador para tu primera transacción

Air Gap

1 Localiza la ramificación "cardano-cli transaction build-raw"



2 Hay que comenzar gradualmente de arriba hacia abajo.

cardano-cli transaction build-raw

```
[ --byron-era
| --shelley-era
| --allegra-era
| --mary-era
| --alonzo-era
| --babbage-era
]
```

3 Los primeros 5 ajustes son opcionales [corchetes] y al no mencionar ajuste, por defecto es era Mary.

cardano-cli transaction build-raw

~~[--byron-era~~
~~| --shelley-era~~
~~| --allegra-era~~
~~| --mary-era~~
~~| --alonzo-era~~
~~| --babbage-era~~
~~]~~
[--script-valid | --script-invalid]

4 Tu transacción no involucra un script por lo que puedes saltar las siguientes 2 opciones.

cardano-cli transaction build-raw

~~[--byron-era~~
~~| --shelley-era~~
~~| --allegra-era~~
~~| --mary-era~~
~~| --alonzo-era~~
~~| --babbage-era~~
~~]~~
~~[--script-valid | --script-invalid]~~

5 Después, para las siguientes opciones y sus 20 subopciones se requiere una explicación.

Dentro de cada corchete puede haber subopciones definidas por columnas. Es por esto que hay una noción de prioridad y un orden particular que debe respetarse al construir una transacción. En este caso sabes que hay 3 columnas distintas que definen el orden en que las opciones se ingresan si queremos que el cuerpo de la transacción se produzca correctamente.

```
1 [~~--script-valid | --script-invalid~~]
  (--tx-in <TX-IN>
2   [--spending-tx-in-reference <TX-IN>
      --spending-plutus-script-v2
3     ( --spending-reference-tx-in-datum-cbor-file
        | --spending-reference-tx-in-datum-file
```

6 Toma tu tiempo para analizar cuidadosamente el orden prioritario de la opción "tx-in" y sus corchetes.

```
~~[--script-valid | --script-invalid]~~
(--tx-in <TX-IN>
 [ --spending-tx-in-reference <TX-IN>
   --spending-plutus-script-v2
   ( --spending-reference-tx-in-datum-cbor-file <CBOR FILE>
   | --spending-reference-tx-in-datum-file <JSON FILE>
   | --spending-reference-tx-in-datum-value <JSON VALUE>
   | --spending-reference-tx-in-inline-datum-present
   )
   ( --spending-reference-tx-in-redeemer-cbor-file <CBOR FILE>
   | --spending-reference-tx-in-redeemer-file <JSON FILE>
   | --spending-reference-tx-in-redeemer-value <JSON VALUE>
   )
   --spending-reference-tx-in-execution-units (<INT, INT>)
 | --simple-script-tx-in-reference <TX-IN>
 | --tx-in-script-file <FILE>
   [
     ( --tx-in-datum-cbor-file <CBOR FILE>
     | --tx-in-datum-file <JSON FILE>
     | --tx-in-datum-value <JSON VALUE>
     | --tx-in-inline-datum-present
     )
     ( --tx-in-redeemer-cbor-file <CBOR FILE>
     | --tx-in-redeemer-file <JSON FILE>
     | --tx-in-redeemer-value <JSON VALUE>
     )
     --tx-in-execution-units (<INT, INT>)]
 ])
[--read-only-tx-in-reference <TX-IN>]
[--tx-in-collateral <TX-IN>]
```

7 "--tx-in" es obligatorio, pero no sus subopciones

Los siguientes ajustes son opcionales y se usan para scripts de Plutus. Por lo que no se deben usar para nuestra simple transacción.

```
--tx-in <TX-IN>
~~[ --spending-tx-in-reference <TX-IN>~~
~~  --spending-plutus-script-v2~~
~~  ( --spending-reference-tx-in-datum-cbor-file <CBOR FILE>~~
~~  | --spending-reference-tx-in-datum-file <JSON FILE>~~
~~  | --spending-reference-tx-in-datum-value <JSON VALUE>~~
~~  | --spending-reference-tx-in-inline-datum-present~~
~~  )~~
~~  ( --spending-reference-tx-in-redeemer-cbor-file <CBOR FILE>~~
~~  | --spending-reference-tx-in-redeemer-file <JSON FILE>~~
~~  | --spending-reference-tx-in-redeemer-value <JSON VALUE>~~
~~  )~~
~~  --spending-reference-tx-in-execution-units (<INT, INT>)~~
~~| --simple-script-tx-in-reference <TX-IN>~~
~~| --tx-in-script-file <FILE>~~
~~  [~~
~~    ( --tx-in-datum-cbor-file <CBOR FILE>~~
~~    | --tx-in-datum-file <JSON FILE>~~
~~    | --tx-in-datum-value <JSON VALUE>~~
~~    | --tx-in-inline-datum-present~~
~~    )~~
~~    ( --tx-in-redeemer-cbor-file <CBOR FILE>~~
~~    | --tx-in-redeemer-file <JSON FILE>~~
~~    | --tx-in-redeemer-value <JSON VALUE>~~
~~    )~~
~~    --tx-in-execution-units (<INT, INT>)]~~
 ])
[--mint <VALUE>
```

8 Por lo tanto:

No necesitas "read only reference input" o collateral ya que es una transacción simple que no incluye un script de Plutus.

```
~~[--read-only-tx-in-reference <TX-IN>]~~
~~[--tx-in-collateral <TX-IN>]~~
~~[--tx-out-return-collateral <ADDRESS VALUE>]~~
~~[--tx-total-collateral <INTEGER>]~~
[--required-signer <FILE> | --required-signer-hash <HASH>]
[--tx-out <ADDRESS VALUE>
  [ --tx-out-datum-hash <HASH>
  | --tx-out-datum-hash-cbor-file <CBOR FILE>
```

9 Acerca de "--required-signer-hash <HASH>"

Esta opción no se usará por ahora en tu transacción para someter tu stake.cert a la cadena de bloques pero nótese que será muy útil para ti en el ejercicio de votación de gobernanza.

```
~~[--read-only-tx-in-reference <TX-IN>]~~
~~[--tx-in-collateral <TX-IN>]~~
~~[--tx-out-return-collateral <ADDRESS VALUE>]~~
~~[--tx-total-collateral <INTEGER>]~~
~~[--required-signer <FILE> | --required-signer-hash <HASH>]~~
[--tx-out <ADDRESS VALUE>
```

10 ¡Finalmente! Una opción que necesitas, "--tx-out".

Necesitas esta opción para especificar la dirección que recibirá el balance de los UTXOs consumidos, menos las comisiones. Hay que copiar esta opción y agregarla al borrador de tu transacción.

```
~~[--read-only-tx-in-reference <TX-IN>]~~
~~[--tx-in-collateral <TX-IN>]~~
~~[--tx-out-return-collateral <ADDRESS VALUE>]~~
~~[--tx-total-collateral <INTEGER>]~~
~~[--required-signer <FILE> | --required-signer-hash <HASH>]~~
--tx-out <ADDRESS VALUE>
```

11 Las subopciones de "tx-out" acerca de scripts de Plutus se pueden omitir. Por el momento, no son necesarias.

```
--tx-out <ADDRESS VALUE>
~~[ --tx-out-datum-hash <HASH>~~
~~| --tx-out-datum-hash-cbor-file <CBOR FILE>~~
~~| --tx-out-datum-hash-file <JSON FILE>~~
~~| --tx-out-datum-hash-value <JSON VALUE>~~
~~| --tx-out-datum-embed-cbor-file <CBOR FILE>~~
~~| --tx-out-datum-embed-file <JSON FILE>~~
~~| --tx-out-datum-embed-value <JSON VALUE>~~
~~| --tx-out-inline-datum-cbor-file <CBOR FILE>~~
~~| --tx-out-inline-datum-file <JSON FILE>~~
~~| --tx-out-inline-datum-value <JSON VALUE>~~
~~ ]~~
~~[--tx-out-reference-script-file <FILE>]]~~
[--mint <VALUE>
```

12 Gradualmente comienzas a comprender. Tus transacciones no son de tipo multi-asset, NFT, ni llevan scripts de Plutus.

```
~~[--mint <VALUE>~~
~~ ( --mint-script-file <FILE>~~
~~   [~~
~~     ( --mint-redeemer-cbor-file <CBOR FILE>~~
~~     | --mint-redeemer-file <JSON FILE>~~
~~     | --mint-redeemer-value <JSON VALUE>~~
~~     )~~
~~    --mint-execution-units (<INT, INT>)]~~
~~ | --simple-minting-script-tx-in-reference <TX-IN> --policy-id <HASH>~~
~~ | --mint-tx-in-reference <TX-IN>~~
~~   --mint-plutus-script-v2~~
~~   ( --mint-reference-tx-in-redeemer-cbor-file <CBOR FILE>~~
~~   | --mint-reference-tx-in-redeemer-file <JSON FILE>~~
~~   | --mint-reference-tx-in-redeemer-value <JSON VALUE>~~
~~   )~~
~~   --mint-reference-tx-in-execution-units (<INT, INT>)~~
~~   --policy-id <HASH>~~
~~ )]~~
```

13 Usarás 3 de las siguientes 4 opciones.

- ''--invalid-before'' determina desde qué Slot será válida la transacción al procesarse.
- mientras ''--invalid-after'' determina desde qué Slot será inválida la transacción. (justo como una fecha de expiración)

```
[--invalid-before <SLOT>]
[--invalid-hereafter <SLOT>]
[--fee <LOVELACE>]
[--certificate-file <CERTIFICATEFILE>
```

14 Obtienes el TTL, la comisión y el archivo del certificado.

Es por esto que hace unos ejercicios ejecutamos "cardano-cli query tip". Al conocer el número de slot de tu nodo sincronizado, puedes determinar un "tiempo de vida" o "TTL" (por sus siglas en inglés) para tu transacción mientras se encuentre en pool de memoria. Por lo que ahora puedes agregar a tu borrador estas tres opciones detalladas más adelante.

```
~~[--invalid-before <SLOT>]~~
[--invalid-hereafter <SLOT>]
[--fee <LOVELACE>]
[--certificate-file <CERTIFICATEFILE>
```

15 De nuevo, no se usan opciones de certificado relacionadas con scripts de Plutus.

```
--certificate-file <CERTIFICATEFILE>
~~[--certificate-script-file <FILE>~~
~~  [~~
~~    ( --certificate-redeemer-cbor-file <CBOR FILE>~~
~~    | --certificate-redeemer-file <JSON FILE>~~
~~    | --certificate-redeemer-value <JSON VALUE>~~
~~    )~~
~~   --certificate-execution-units (<INT, INT>)]~~
~~ | --certificate-tx-in-reference <TX-IN>~~
~~   --certificate-plutus-script-v2~~
~~   ( --certificate-reference-tx-in-redeemer-cbor-file <CBOR FILE>~~
~~   | --certificate-reference-tx-in-redeemer-file <JSON FILE>~~
~~   | --certificate-reference-tx-in-redeemer-value <JSON VALUE>~~
~~   )~~
~~   --certificate-reference-tx-in-execution-units (<INT, INT>)~~
~~ ]]~~
```

16 **La opción "--withdrawal" es una entrada que permite retirar las recompensas en stake.ad**

```
[--withdrawal <WITHDRAWAL>
 [ --withdrawal-script-file <FILE>
   [
    ( --withdrawal-redeemer-cbor-file <CBOR FILE>
    | --withdrawal-redeemer-file <JSON FILE>
    | --withdrawal-redeemer-value <JSON VALUE>
    )
    --withdrawal-execution-units (<INT, INT>)]
 | --withdrawal-tx-in-reference <TX-IN>
   --withdrawal-plutus-script-v2
   ( --withdrawal-reference-tx-in-redeemer-cbor-file <CBOR FILE>
   | --withdrawal-reference-tx-in-redeemer-file <JSON FILE>
   | --withdrawal-reference-tx-in-redeemer-value <JSON VALUE>
   )
   --withdrawal-reference-tx-in-execution-units (<INT, INT>)
 ]]
[--json-metadata-no-schema | --json-metadata-detailed-schema]
[--auxiliary-script-file <FILE>]
[--metadata-json-file <FILE> | --metadata-cbor-file <FILE>]
[--genesis <FILE> | --protocol-params-file <FILE>]
[--update-proposal-file <FILE>]
--out-file <FILE>
```

17 **Puedes omitir "--withdrawal" y aquellas relacionadas con scripts de Plutus por ahora.**

```
[--withdrawal <WITHDRAWAL>
 [ --withdrawal-script-file <FILE>
   [
    ( --withdrawal-redeemer-cbor-file <CBOR FILE>
    | --withdrawal-redeemer-file <JSON FILE>
    | --withdrawal-redeemer-value <JSON VALUE>
    )
    --withdrawal-execution-units (<INT, INT>)]
 | --withdrawal-tx-in-reference <TX-IN>
   --withdrawal-plutus-script-v2
   ( --withdrawal-reference-tx-in-redeemer-cbor-file <CBOR FILE>
   | --withdrawal-reference-tx-in-redeemer-file <JSON FILE>
   | --withdrawal-reference-tx-in-redeemer-value <JSON VALUE>
   )
   --withdrawal-reference-tx-in-execution-units (<INT, INT>)
 ]]
[--json-metadata-no-schema | --json-metadata-detailed-schema]
[--auxiliary-script-file <FILE>]
[--metadata-json-file <FILE> | --metadata-cbor-file <FILE>]
[--genesis <FILE> | --protocol-params-file <FILE>]
[--update-proposal-file <FILE>]
--out-file <FILE>
```

18 **Sólo hay pocas opciones restantes.**

No tienes metadatos a someter por ahora, ni un archivo script auxiliar, ni es necesario especificar un génesis o archivos de parámetros del protocolo. No debes someter una actualización de propuesta para el fondo Catalyst. Sólo queda nombrar el archivo para tu borrador de transacción. (--out-file <FILE>).

```
[--json-metadata-no-schema | --json-metadata-detailed-schema]
[--auxiliary-script-file <FILE>]
[--metadata-json-file <FILE> | --metadata-cbor-file <FILE>]
[--genesis <FILE> | --protocol-params-file <FILE>]
[--update-proposal-file <FILE>]
--out-file <FILE>
```

19 **Al agrupar las opciones que copiaste durante este ejercicio, obtendrás algo similar a:**

cardano-cli transaction build-raw

```
--tx-in <TX-IN>
--tx-out <ADDRESS VALUE>
[--invalid-hereafter <SLOT>]
[--fee <LOVELACE>]
[--certificate-file <CERTIFICATEFILE>
--out-file <FILE>
```

20 **Ahora para completar tu borrador:**

Puedes agregar la entrada UTXO (tx-in), la dirección para el cambio (tx-out) y el archivo de certificado. Por ahora, asignamos un valor de 0 a tx-out, invalid-hereafter y fee.

cardano-cli transaction build-raw

```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+0
--invalid-hereafter 0
--fee 0
--certificate-file stake.cert
--out-file tx.raw
```

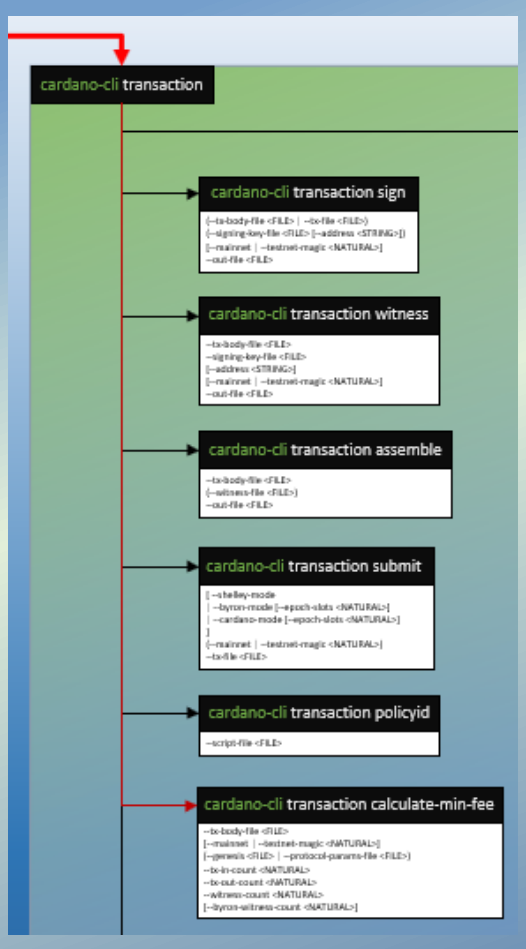
21 **Así es como se debe ver en tu terminal:**

```
user@computer:~$ cardano-cli transaction build-raw \
> --tx-in 1234a4d18e9dkhb34234kjbvd81e299c#0 \
> --tx-out $(cat paymentwithstake.addr)+0 \
> --invalid-hereafter 0 \
> --fee 0 \
> --certificate-file stake.cert \
> --out-file tx.raw
```

¡Felicidades! Llegaste hasta aquí. Guarda el comando y opciones de 21 en un archivo en cualquier editor de texto, ya que lo necesitarás después del siguiente ejercicio. Ahora vas a calcular las comisiones incurridas por tu transacción. Ahora vas a restarla de la cantidad de tu UTXO (tx-in) y no olvides incluir el depósito para el registro de una dirección de stake.

Noveno ejercicio: Cálculo de las comisiones — Air Gap

1 **Localiza la ramificación que usarás para calcular la comisión.**



2 **Hay 9 opciones en total.**

Este comando te dará la cantidad exacta de las comisiones que debes pagar, dependiendo del número de tx-in, tx-out y el número de firmas requeridas.

cardano-cli transaction calculate-min-fee

```
--tx-body-file <FILE>
[--mainnet | --testnet-magic <NATURAL>]
(--genesis <FILE> | --protocol-params-file <FILE>)
--tx-in-count <NATURAL>
--tx-out-count <NATURAL>
--witness-count <NATURAL>
[--byron-witness-count <NATURAL>]
```

3 **Sólo 3 de estas opciones no se usarán.**

- testnet-magic (obviamente usaremos mainnet en este tutorial)
- genesis (usa los parámetros del protocolo obtenidos anteriormente)
- byron-witness-count (porque no usas pares de llaves era Byron)

cardano-cli transaction calculate-min-fee

```
--tx-body-file <FILE>
[--mainnet | --testnet-magic <NATURAL>]
(--genesis <FILE> | --protocol-params-file <FILE>)
--tx-in-count <NATURAL>
--tx-out-count <NATURAL>
--witness-count <NATURAL>
[--byron-witness-count <NATURAL>]
```

4 **Sólo se usan 3 de estas opciones**

Hay que especificar el número de direcciones de entrada y salida, así como el número de llaves usadas para firmar tu transacción.

cardano-cli transaction calculate-min-fee

```
--tx-body-file <FILE>
[--mainnet | --testnet-magic <NATURAL>]
(--genesis <FILE> | --protocol-params-file <FILE>)
--tx-in-count 1
--tx-out-count 1
--witness-count 2
[--byron-witness-count <NATURAL>]
```

5 **Después debes indicar la ruta PATH a tu archivo protocol.json y a tu transacción draft tx.raw**

cardano-cli transaction calculate-min-fee

```
--tx-body-file tx.raw
--mainnet
--protocol-params-file protocol.json
--tx-in-count 1
--tx-out-count 1
--witness-count 2
```

6 **Este es el resultado en tu terminal. (La cantidad de las comisiones no siempre es la misma.)**

```
user@computer:~$ cardano-cli transaction calculate-min-fee \
> --tx-body-file tx.raw \
> --mainnet \
> --protocol-params-file protocol.json \
> --tx-in-count 1 \
> --tx-out-count 1 \
> --witness-count 2
```

Si el comando funciona como se pretende, las comisiones deben aparecer en la parte inferior.

```
user@computer:~$ cardano-cli transaction calculate-min-fee \
> --tx-body-file tx.raw \
> --mainnet \
> --protocol-params-file protocol.json \
> --tx-in-count 1 \
> --tx-out-count 1 \
> --witness-count 2
178525 Lovelace
```

Para el siguiente ejercicio debes abrir el archivo en el editor de texto que usaste para guardar el comando "cardano-cli transaction build-raw" del ejercicio ocho. Vas a modificar su contenido para construir tu transacción final.

Décimo ejercicio: Construir la transacción final — Air Gap

1 Este es tu borrador de transacción del ejercicio ocho.

Modifica el borrador para ingresar la cantidad de las comisiones (que ahora ya conoces) y calcula el monto en Lovelace que se envía de vuelta a tu dirección.

cardano-cli transaction build-raw

```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+0
--invalid-hereafter 0
--fee 178525
--certificate-file stake.cert
--out-file tx.raw
```

2 Puedes llevar calcularlo usando el comando "expr"

Monto de UTXO — Depósito a dirección de stake — comisión

```
user@computer:~$ expr 10000000 - 178525 – 2000000
```

```
user@computer:~$ expr 10000000 - 178525 – 2000000
7821475
user@computer:~$
```

3 Puedes agregar el resultado a tu transacción

Nótese que no debe haber espacio entre tu dirección, el operador "+" y la cantidad en Lovelace. De otra forma, habrá un error al ejecutar el comando.

cardano-cli transaction build-raw

```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+7821475
--invalid-hereafter 0
--fee 178525
--certificate-file stake.cert
--out-file tx.raw
```

4 Ahora, determina el "TTL" (time-to-live)

Para escoger desde qué Slot será inválida la transacción, debes conocer el número de Slot presente, repitiendo el ejercicio #6 o revisando los logs. Aquí hay un ejemplo de lo que puedes obtener:

```
{
  "block": 8749178,
  "epoch": 410,
  "era": "Babbage",
  "hash": "367e4af96abc18e1d4b5de08af535cb508e691...",
  "slot": 92029934,
  "syncProgress": "100.00"
}
```

5 Agrega unos minutos (1 slot = 1 segundo)

De forma que tengas tiempo para firmar la transacción y someterla en el nodo sincronizado "hot node", agrega 15 minutos al valor de la opción. (92029934 + 900 = 92030834)

cardano-cli transaction build-raw

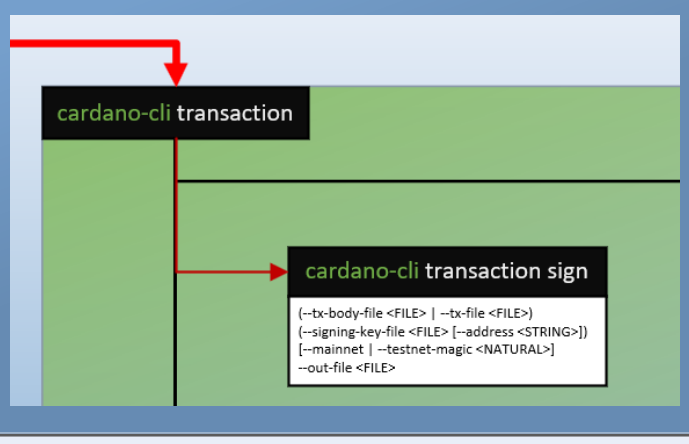
```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+7821475
--invalid-hereafter 92030834
--fee 178525
--certificate-file stake.cert
--out-file tx.raw
```

6 Este es el resultado en tu terminal:

```
user@computer:~$ cardano-cli transaction build-raw \
> --tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0 \
> --tx-out $(cat paymentwithstake.addr)+7821475 \
> --invalid-hereafter 92030834 \
> --fee 178525 \
> --certificate-file stake.cert \
> --out-file tx.raw
```

Onceavo ejercicio: Firmar tu transacción — Air Gap

1 Ahora puedes firmar la transacción con tus 2 llaves privadas (payment.skey y stake.skey)



2 Hay un total de 7 opciones.

Para esto, debes mencionar tu "archivo de cuerpo para transacción", ruta PATH de tus 2 llaves privadas, la red a usar y el nombre del archivo que quieres someter a la cadena de bloques.

cardano-cli transaction sign

```
(--tx-body-file <FILE> | --tx-file <FILE>)
(--signing-key-file <FILE> [--address <STRING>])
[--mainnet | --testnet-magic <NATURAL>]
--out-file <FILE>
```

3 Este es el resultado en tu terminal:

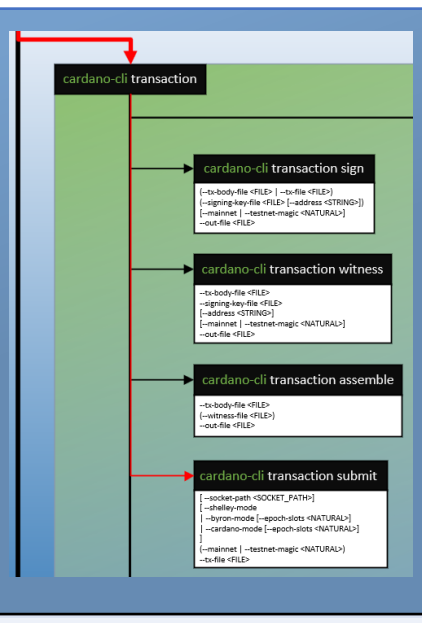
```
user@computer:~$ cardano-cli transaction sign\
> --tx-body-file tx.raw \
> --signing-key-file payment.skey \
> --signing-key-file stake.skey \
> --mainnet \
> --out-file tx.signed
```

⚠ **Nótese que en muchas situaciones, algunas opciones se usan múltiples veces.**

Ahora puedes transferir "tx.signed" a tu "Hot Node" para someterlo a la cadena de bloques, pero antes asegúrate de que los permisos del archivo sean únicamente de lectura "Read-only"

Doceavo ejercicio: Somete tu transacción — Hot Node

1 ¡Tu transacción está lista para ejecutarse!



2 Tienes 9 opciones.

El ajuste "--socket-path" es opcional si la ruta (PATH) de tu node.socket ya está en tu entorno. Sólo usa los ajustes requeridos. En otras palabras, la red y el nombre del archivo a someter.

cardano-cli transaction submit

```
[ --socket-path <SOCKET_PATH>]
[ --shelley-mode
| --byron-mode [ --epoch-slots <NATURAL>]
| --cardano-mode [ --epoch-slots <NATURAL>]
]
(--mainnet | --testnet-magic <NATURAL>)
--tx-file <FILE>
```

3 Este es el comando en la terminal:

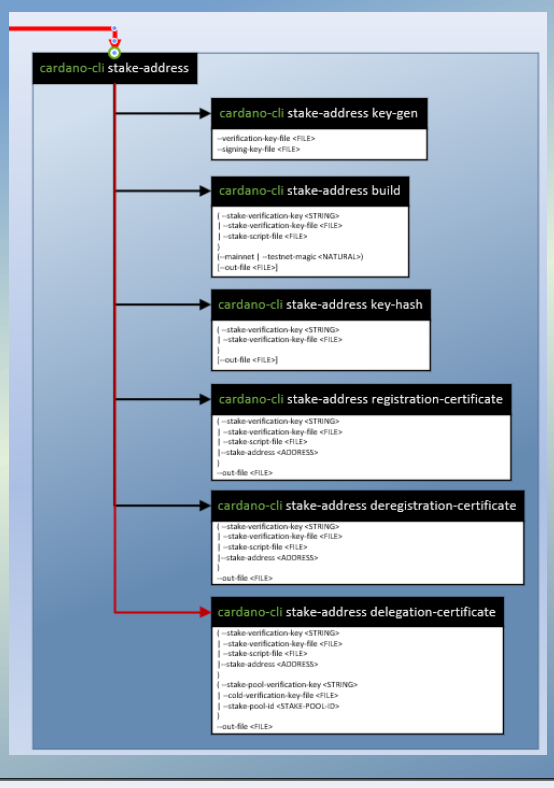
```
user@computer:~$ cardano-cli transaction submit \
> --mainnet \
> --tx-file tx.signed
```

```
user@computer:~$ cardano-cli transaction submit \
> --mainnet \
> --tx-file tx.signed
transaction successfully submitted
```

Felicidades, tu dirección de stake ya está registrada en la cadena de bloques. Ahora puedes crear un certificado de delegación para escoger una stake pool y participar en el protocolo de Prueba de Participación "Proof of Stake" de Cardano. Sin embargo, antes de seguir al próximo ejercicio, asegúrate de borrar el archivo tx.signed de tu nodo sincronizado "Hot Node". (Ya no es necesario)

Treceavo ejercicio: Creación de un certificado de delegación — Air Gap

1 Primero, localiza la ramificación donde vas a usar tu certificado.



2 Hay 8 opciones en total.

Una vez sometido en la cadena de bloques, este certificado se usará para indicar a qué stake pool está delegada tu ADA. Hay 2 grupos de ajustes requeridos. Estos ajustes se eligen dependiendo de tus necesidades (por ejemplo, usar cold.vrf para tu propia stake pool). En este ejercicio, asumimos que quieres elegir entre una de las stake pools.

cardano-cli stake-address delegation-certificate

```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
|--stake-address <ADDRESS>
)
( --stake-pool-verification-key <STRING>
| --cold-verification-key-file <FILE>
| --stake-pool-id <STAKE-POOL-ID>
)
--out-file <FILE>
```

3 Usemos –stake-address y –stake-pool-id

- Especifica la ruta (PATH) de tu stake.addr.
- El identificador (ID) de la stake pool a la que delegaste tu cartera. (puede ser codificado en Bech32 o en Hex).
- El nombre del archivo del certificado.

cardano-cli stake-address delegation-certificate

```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
  --stake-address stake.addr

( --stake-pool-verification-key <STRING>
| --cold-verification-key-file <FILE>
  --stake-pool-id pool1mt8sdg37f2h3rypyuc77k7vxrjshtvjw04zdjlae9vdzyt9uu34
)
--out-file delegation.cert
```

4 Este es el resultado en tu terminal:

```
user@computer:~$ cardano-cli stake-address delegation-certificate \
> --stake-address stake.addr \
> --stake-pool-id pool1mt8sdg37f2h3rypyuc77k7vxrjshtvjw04zdjlae9vdzyt9uu34 \
> --out-file delegation.cert
```

Puedes obtener el ID de la stake pool en cexplorer.io o si te agrada este documento, haznos saber, podemos agregar comandos: "query stake-pools", "query pool-state" y "query pool-distribution" a el 2ndo capítulo de este tutorial.

Ahora puedes repetir los ejercicios 6 al 12, reemplazando stake.cert con delegation.cert al construir la transacción. No olvides que al calcular las comisiones no debes tomar en cuenta el depósito a la dirección de stake. (que ya se hizo previamente)

Resumen de operaciones: Proceso para someter un certificado de delegación

Air Gap (Fuera de línea)

cardano-cli transaction build-raw

```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+0
--invalid-hereafter 0
--fee 0
--certificate-file delegation.cert
--out-file tx.raw
```

cardano-cli transaction calculate-min-fee

```
--tx-body-file tx.raw
--mainnet
--protocol-params-file protocol.json
--tx-in-count 1
--tx-out-count 1
--witness-count 2
```

cardano-cli transaction build-raw

```
--tx-in 4321a4d18e9dkhb34234kjbvdec3ad81e654g#0
--tx-out $(cat paymentwithstake.addr)+7821475
--invalid-hereafter 92040794
--fee 176625
--certificate-file delegation.cert
--out-file tx.raw
```

cardano-cli transaction sign

```
--tx-body-file tx.raw
--signing-key-file payment.skey
--signing-key-file stake.skey
--mainnet
--out-file tx.signed
```

Hot Node (En línea)

cardano-cli query tip

```
--mainnet
```

cardano-cli query utxo

```
--address paymentwithstake.addr
--mainnet
--out-file UTXO.addrs
```

cardano-cli transaction submit

```
--mainnet
--tx-file tx.signed
```

Concluimos la parte 1 de este tutorial con una cita de un colega operador (SPO) que aprecio mucho:

"Debemos alentar a los nuevos SPOs, incluso con habilidades (técnicas) bajas. Ellos aprenderán y se descentralizará Cardano" - @StakeWithPride